Package 'tfNeuralODE'

May 8, 2026

Type Package

Title Create Neural Ordinary Differential Equations with 'tensorflow'

Version 0.1.0

Maintainer Shayaan Emran <shayaan.emran@gmail.com>

Description Provides a framework for the creation and use of Neural ordinary differential equations with the 'tensorflow' and 'keras' packages. The idea of Neural ordinary differential equations comes from Chen et al. (2018) <doi:10.48550/arXiv.1806.07366>, and presents a novel way of learning and solving differential systems.

License MIT + file LICENSE

Encoding UTF-8

Imports tensorflow, keras, reticulate, deSolve

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/semran9/tfNeuralODE

BugReports https://github.com/semran9/tfNeuralODE/issues

NeedsCompilation no

Author Shayaan Emran [aut, cre, cph]

Repository CRAN

Date/Publication 2023-10-16 17:30:02 UTC

Contents

backward *Backward pass of the Neural ODE*

Description

Backward pass of the Neural ODE

Usage

```
backward(model, tsteps, outputs, output_gradients = NULL)
```

Arguments

model	A keras neural network that defines the Neural ODE.
tsteps	A vector of each time step upon which the Neural ODE is solved to get to the final solution.
outputs	The tensor outputs of the forward pass of the Neural ODE.
output_gradients	The tensor gradients of the loss function.

Value

The model input at the last time step.

The gradient of loss with respect to the inputs for use with the Adjoint Method.

The gradients of loss the neural ODE.

Examples

```
reticulate::py_module_available("tensorflow")

# example code
# single training example
OdeModel(keras$Model) %py_class% {
 initialize <- function() {
   super$initialize()
   self$block_1 <- layer_dense(units = 50, activation = 'tanh')
   self$block_2 <- layer_dense(units = 2, activation = 'linear')
 }

 call <- function(inputs) {
   x<- inputs ^ 3
   x <- self$block_1(x)
   self$block_2(x)
 }
}
tsteps <- seq(0, 2.5, by = 2.5/10)
true_y0 = t(c(2., 0.))
```

```
model<- OdeModel()
optimizer = tf$keras$optimizers$legacy$Adam(learning_rate = 1e-3)
# single training iteration
pred = forward(model, true_y0, tsteps)
with(tf$GradientTape() %as% tape, {
  tape$watch(pred)
  loss = tf$reduce_mean(tf$abs(pred - inp[[2]]))
})
dLoss = tape$gradient(loss, pred)
list_w = backward(model, tsteps[1:batch_time], pred, output_gradients = dLoss)
optimizer$apply_gradients(zip_lists(list_w[[3]], model$trainable_variables))
```

`forward` *Forward pass of the Neural ODE network*

Description

Forward pass of the Neural ODE network

Usage

```
forward(model, inputs, tsteps, return_states = FALSE)
```

Arguments

`model`	A keras neural network that defines the Neural ODE.
`inputs`	Matrix or vector inputs to the neural network.
`tsteps`	A vector of each time step upon which the Neural ODE is solved to get to the final solution.
`return_states`	A boolean which dictates whether the intermediary states between the input and the final solution are returned.

Value

solution of the forward pass of Neural ODE

Examples

```
reticulate::py_module_available("tensorflow")

# example code

library(tensorflow)
library(keras)

OdeModel(keras$Model) %py_class% {
 initialize <- function() {
```

```
    super$initialize()
    self$block_1 <- layer_dense(units = 50, activation = 'tanh')
    self$block_2 <- layer_dense(units = 2, activation = 'linear')
  }

  call <- function(inputs) {
    x<- inputs ^ 3
    x <- self$block_1(x)
    self$block_2(x)
  }
}
tsteps <- seq(0, 2.5, by = 2.5/10)
true_y0 = t(c(2., 0.))
model<- OdeModel()
forward(model, true_y0, tsteps)
```

---

rk4_step *Runge Kutta solver for ordinary differential equations*

Description

Runge Kutta solver for ordinary differential equations

Usage

```
rk4_step(func, dt, state)
```

Arguments

func	The function to be numerically integrated.
dt	Time step.
state	A list describing the state of the function, with the first element being 1, and the second being a tensor that represents state

Value

A list containing a new time and the numerical integration of of the function across the time step to the new time.

Examples

```
reticulate::py_module_available("tensorflow")
# example code
library(tensorflow)
ode_fun<- function(u){
  r = u ^ 3
  true_A = rbind(c(-0.1, 2.0), c(-2.0, -0.1))
```

```
  du <- r %*% (true_A)
  return(as.matrix(du))
}
y<- tensorflow::tf$cast(t(as.matrix(c(2, 0))), dtype = tf$float32)
x<- rk4_step(ode_fun,  dt = 0.25,
             state = list(1.0, y))
x
```

Index